
WPI

- TI - Higher order synthesis method for architectural design of LSI - involves creating new control flow graph and data dependence graph automatically using computer without waiting for completion of previous loop
- AB - J08044773 The method involves using a computer to prepare a control flow graph and a data dependence graph. The output variables in a particular loop which are alike and which affect a following loop, are specified. Using the specified values, a new control flow graph and data dependence graph are created automatically without waiting for the completion of the previous loop.
- ADVANTAGE - Obtains LSI with quick processing speed. Obtains highly efficient pipeline structure by using control flow graph and data dependence graph.
- (Dwg.3/20)
- PN - JP8044773 A 19960216 DW199617 G06F17/50 024pp
- PR - JP19940174368 19940726
- PA - (MITQ) MITSUBISHI ELECTRIC CORP
- MC - T01-J15
- DC - T01
- IC - G06F17/50
- AN - 1996-164325 [17]

PAJ

- TI - AUTOMATIC HIGH-LEVEL COMPOSING METHOD
- AB - PURPOSE: To provide the high-level composing method which automatically composes the architecture of a high-performance LSI with pipeline structure.
- CONSTITUTION: In a loop description, an output variable which affects a next loop is specified by the arithmetic of the loop (processes S12 and S13) and arithmetic related to the specified output variable is specified (process S14 and S15). After arithmetic is classified into two groups (processes S16 and S17) according to the data dependency relation between the loops, a control flow graph and a data dependency graph are newly generated again. The newly generated control flow graph and data dependency graph are used to compose the architecture having the pipeline structure.
- PN - JP8044773 A 19960216
- PD - 1996-02-16
- ABD - 19960628
- ABV - 199606
- AP - JP19940174368 19940726
- PA - MITSUBISHI ELECTRIC CORP
- IN - HIGASHIDA MOTOKI
- I - G06F17/50

THIS PAGE BLANK (USPTO)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-44773

(43) 公開日 平成8年(1996)2月16日

(51) IntCl.⁴

G 0 6 F 17/50

識別記号

庁内整理番号

F I

技術表示箇所

9191-5H

G 0 6 F 15/ 60

3 6 0 A

審査請求 未請求 請求項の数 11 O L (全 24 頁)

(21) 出願番号

特願平6-174368

(22) 出願日

平成6年(1994)7月26日

(71) 出願人 000006013

三菱電機株式会社

東京都千代田区丸の内二丁目2番3号

(72) 発明者 東田 基樹

兵庫県伊丹市瑞原4丁目1番地 三菱電機

株式会社システムエル・エス・アイ開発研

究所内

(74) 代理人 弁理士 吉田 茂明 (外2名)

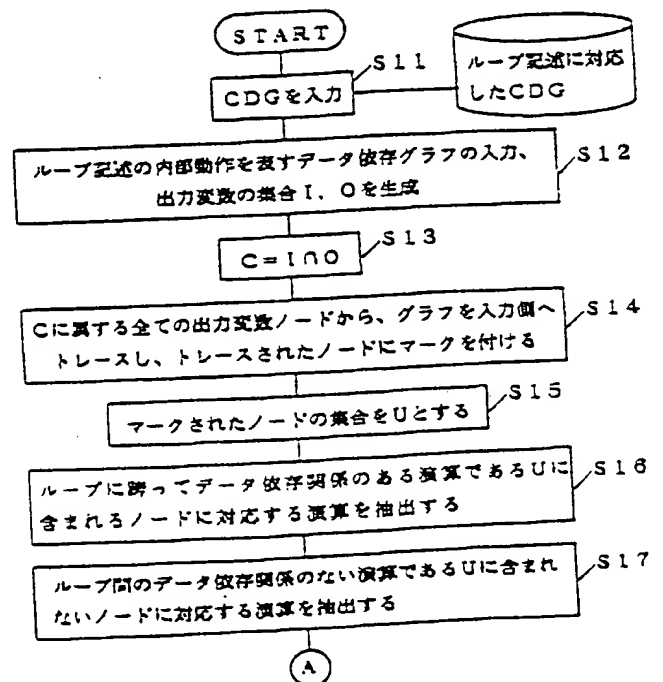
(54) 【発明の名称】 自動高位合成方法

(57) 【要約】

【目的】 バイブライン構造を有する高性能なLSIのアーキテクチャを自動合成する高位合成方法を得ることを目的とする。

【構成】 ループ記述のうち、そのループでの演算によって次のループに影響を与える出力変数を特定し(工程S12、S13)、特定された出力変数に関わる演算を特定する(工程S14、S15)。そして、ループ間のデータ依存関係によって、2つのグループに分類した後(工程S16、S17)、新たに、コントロールフローグラフとデータ依存グラフとを再度生成する。

【効果】 新たに生成されたコントロールフローグラフとデータ依存グラフを用いることによって、バイブライン構造を持ったアーキテクチャの合成が可能になる。



1

【特許請求の範囲】

【請求項1】 ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程と、
計算機を用いて、自動的に、前記ループ記述に対応したコントロールフローグラフ及び前記データ依存グラフを、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程とを備える、自動高位合成方法。

【請求項2】 前記変形工程は、
前記ループ記述の内部動作を表す前記データ依存グラフの入力変数を要素とする第1の集合と、前記データ依存グラフの出力変数を要素とする第2の集合とを生成する工程と、
前記第1の集合と前記第2の集合との積集合を生成する工程と、
前記積集合に属する出力変数を出力するノードから、前記データ依存グラフを入力側へトレースして、前記出力変数を出力するために必要な演算を行う第1のノードとそれ以外の演算を行う第2のノードとに分離抽出する工程と、
前記新たなデータ依存グラフとして、前記第1のノードからなる第1のデータ依存グラフと前記第2のノードからなる第2のデータ依存グラフとを生成するとともに、前記新たなコントロールフローグラフとして、前記第1及び第2のデータ依存グラフの実行手順を規定するコントロールフローグラフを生成する工程とを含む、請求項1記載の自動高位合成方法。

【請求項3】 前記計算機に準備した演算器のデータの中から、前記新たなデータ依存グラフ中の全ての前記ノードにそれぞれ一対一に対応する演算器のデータを関連づけるバインディング工程と、
前記計算機において、前記演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための前記演算器の制御論理及びデータバス構造を自動生成する生成工程とをさらに備える、請求項1または請求項2記載の自動高位合成方法。

【請求項4】 前記生成工程は、
前記ループの脱出条件を判定するノードを第1のステップに割り付ける第1のスケジューリング工程と、
前記第1のノードに対応する前記演算器について、前記第1のデータ依存グラフの制約に基づいて、第2のステップ以降に順に割り付け、制御ステップの決定を行う第2のスケジューリング工程と、
前記第2のノードに対応する前記演算器について、前記第2のデータ依存グラフの制約に基づいて、制御ステップの決定を行う第3のスケジューリング工程と、
接続すべき前記演算器の出力と入力の前記制御ステップが異なっているときには、その差異段数と等しい数のパイプラインレジスタを挿入した後に、差異段数がないと

2

きにはそのまま通常のリソース接続を行うリソース接続工程とを備える、請求項3記載の自動高位合成方法。

【請求項5】 前記演算器の遅延値が1クロック周期以上の場合に、前記演算器をラッチで分離されたパイプライン演算器としてモデル化する工程をさらに備える、請求項3または請求項4記載の自動高位合成方法。

【請求項6】 前記生成工程は、遅延値が1クロック周期以上の前記演算器に対応して一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やす工程を含むことを特徴とする、請求項3ないし請求項5のいずれかに記載の自動高位合成方法。

【請求項7】 前記生成工程は、一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やすとともに複数の前記ノードにおいて一部の演算器を共有させる工程を含むことを特徴とする、請求項3ないし請求項6のいずれかに記載の自動高位合成方法。

【請求項8】 前記新たなコントロールフローグラフ及びデータ依存グラフによって表される回路の一部のパイプラインレジスタの占有面積と前記パイプラインレジスタによって与えられる値を出力する所定の演算を行う回路の占有面積とを比較する比較工程と、
前記比較工程によって、前記所定の演算を行う回路を用いる方が回路面積を小さくできると判断された場合に、前記パイプラインレジスタを前記所定の演算を行う回路に置換する置換工程とをさらに備える、請求項3ないし請求項7のいずれかに記載の自動高位合成方法。

【請求項9】 前記ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程は、
前記コントロールフローグラフ及びデータ依存グラフで表すべき回路を表現した動作レベルのハードウェア記述言語による記述を入力する工程を含み、
前記変形工程で変形された前記新たなコントロールフローグラフ及びデータ依存グラフを表現した、論理合成ツールが直接処理可能なレジスタトランスフェレレベルのハードウェア記述言語による記述を出力する工程をさらに備える、請求項1ないし請求項8のいずれかに記載の自動高位合成方法。

【請求項10】 複数の演算によって構成される一つの関数を表す新たな演算器を追加登録する工程と、
前記新たな演算器を他の前記演算器と同様に処理するために、前記動作レベルのハードウェア記述言語で特殊なコメントを挿入する工程と、をさらに備える、請求項9記載の自動高位合成方法。

【請求項11】 前記ループ記述で表現されたループのうち複数回分のループを、1回分のループとして表現するループ記述に自動展開する工程をさらに備える、請求項9記載の自動高位合成方法。

【発明の詳細な説明】

【0001】

50

【産業上の利用分野】この発明は、LSIのアーキテクチャ設計に用いられる高位合成方法に関し、特にループを含んだ記述から、パイプライン構造をもった高性能なLSIのアーキテクチャを自動合成する高位合成方法に関するものである。

【0002】

【従来の技術】図16は、従来の高位合成方法の手順を示すフローチャートである。数1に示す1つのFORループを含んだ入力記述を例にして、図16に沿って従来の高位合成方法を説明する。

【0003】

【数1】

```
for(i=0; i<N; i=i+1)begin
  ADR=i+base;      /*set address*/
  Data=Input(ADR); /*input from a port*/
  Res=Data*Coef;    /*calculation*/
  output(ADR,Res); /*output to a port*/
end
```

【0004】はじめに、工程S31において、準備された動作レベルのハードウェア記述言語による入力記述をアセンブラコードのようなプリミティブな演算の系列へと変換する。続いて、工程S32において、その系列から、コントロールフローグラフとデータ依存グラフを生成する。ソフトウェアでは、通常、先に記述されたステートメントから逐次実行される。しかし、ハードウェアによる実行では、これらのステートメントを可能ならば、並列に実行することをコンピュータが考慮する。コントロールフローグラフとデータ依存グラフは、この並列実行の可能性を表現している。

【0005】図17は、工程S32において生成された、数1の記述に対応するコントロールフローグラフとデータ依存グラフである。図17において、1は数1のループ記述に対応するループ変数の初期化($i=0$)を行うブロック、2は数1のループ記述に対応する脱出条件の比較($i<N$)を行うブロック、3は数1のループの内部動作及びカウンタのインクリメント($i=i+1$)を行うブロックである。このように数1のループ記述は、3つのブロック1～3に分割できる。

*【0006】数1に示したFORループに対応した処理を実行するためには、ブロック1を始めに実行し、続いてブロック2、ブロック3を交互に実行する必要がある。この実行順を破線のコントロールフローグラフが表現している。一方、各ブロックの内部動作については、変数の依存関係が各演算の実行順序に制約を与える。この制約を表したものが、データ依存グラフである。例えば、output(ADR,Res)という演算がノードE6で行なわれるためには、2つの変数ADR,Resの値が必要である。この時、変数ADRを求めるための演算($ADR=i+base$)を行う加算演算のノードE3と、変数Resを求めるための演算($Res=Data*Coef$)を行う乗算演算のノードE5から、実線の枝がひかれる。枝の存在は、それらの演算にデータ依存関係があることを意味し、それらを並列(同時)に動作させることはできない。

【0007】一方、ノードE2、E3の加算演算($i=i+1$, $ADR=i+base$)は、互いに依存関係がないので、並列に実行可能である。高位合成方法では、はじめに、入力記述をこのようなコントロールフローグラフとデータ依存グラフへ表現し直し、以降の高位合成の処理は、これらのグラフを基にして進める。以後、コントロールフローグラフとデータ依存グラフをCDGと呼ぶ。

【0008】続いて、工程S35において、CDGに基づいてリソースバインディング、スケジューリング、リソース接続、制御論理の生成が行なわれる。リソースバインディングでは、データ依存グラフ上の各ノード(演算)を実行する演算器(リソース)を決定する。スケジューリングでは、CDGにより規制された演算の実行順の範囲内で、必要な制御ステップ数やレジスタ数や接続の複雑度等を考慮して、演算器を動作させるべき制御ステップを決定する。リソース接続では、必要に応じて、セクタやレジスタを挿入しながら、リソース間の接続を行なう。制御論理の生成では、スケジューリング結果を状態遷移として表現するとともに、適当な状態(制御ステップ)で、演算器やレジスタ、セクタに制御信号を出力するような制御論理を生成する。

【0009】

* 【表1】

ノード	演算	バインドした演算器の種類	演算器名
E1	<	LSS	LSS
E2	+	ADD1	ADD1-1
E3	+	ADD1	ADD1-2
E4	input	INP	INP
E5	*	MLP1	MLP1
E6	output	OUT1	OUT

【0010】表1に、リソースバインディングの一例を示す。ここでは、各ノードに、1つの異なる演算器を割り付けた。例えば、ノードE1にはデータの大小を比*

*較する演算を行う演算器LSSを割り付けた。

【0011】

【表2】

演算器名	LSS	ADD1-1	ADD1-2	INP	MLP1	OUT
ステップ0	E1					
ステップ1		E2	E3			
ステップ2				E4		
ステップ3					E5	
ステップ4						E6

【0012】また、表1のリソースバインディングに対する、スケジューリングの一例を表2に示す。このスケジューリングでは、各演算器の遅延時間は1クロック周期以内であると仮定している。図18は、このようなリソースバインディングとスケジューリングに対してリソ※

※ース接続を行なった結果得られたデータバス構造を示すブロック図である。

【0013】

【表3】

現状態	演算器名			レジスタ				次状態
	INP	OUT	セクタ	i	Adr	Data	Res	
ステート0			select 2	WRITE				ステート0
ステート1								ステート2 (S1=1) ステート6 (S1=0)
ステート2			select 1	WRITE	WRITE			ステート3
ステート3	active					WRITE		ステート4
ステート4							WRITE	ステート5
ステート5		active						ステート1
ステート6								ステート6

【0014】また、表3が、制御論理生成の結果である。図18は、LSIのデータバス構造を、表3はLSIを制御するための状態遷移を表現している。図18、及び表3により、数1の入力記述を実現するLSIのアーキテクチャが完成している。以上の処理で、LSIのアーキテクチャの合成ができる。

【0015】最後に、工程S36において、合成されたアーキテクチャを論理合成ツールに直接入力可能なレジスタ・トランスファ・レベル(RTL)のハードウェア記述言語による記述として出力して、高位合成が終了する。

【0016】

【発明が解決しようとする課題】以上のように、従来の高位合成方法で合成された図18及び表3のアーキテクチャでは、数1の入力記述のループがN回まわるにも関わらず、1回のループを5ステップで実行し、全動作が★50

★(5×N+1)ステップで終了する。従来の高位合成方法では、これ以上、高性能なアーキテクチャ(5×Nステップ以下のステップ数で処理できるアーキテクチャ)を合成することはできないという問題点があった。

【0017】しかし、数1の動作記述を実現する、より高性能なアーキテクチャは存在する。図19に、動作終了までの、図18のデータバス構造の各演算器の動作状況を示す。図19において、92で示した列はステップを表しており、93で示した行は演算器の名称を表している。これを見ると分かるように、各演算器は、5ステップに1度しか動作せず、残りのステップでは動作していないことが分かる。例えば、演算器LSSは、ステップ0でi=0の処理をし、ステップ5でi=1の処理をし、ステップ10でi=2の処理をするなど、5ステップ毎に処理が行っている。この空きをなくして異なるループの動作をオーバーラップさせて、パイプライン状に

7

実行させることにより、全動作をN+4ステップで実行できるアーキテクチャが存在する。このようなアーキテクチャをパイプラインアーキテクチャと呼ぶ。

【0018】従来の高位合成方法では、ループ動作は、図17のようなCDGに変換される。このCDGは、ループの内部動作の終了を待ってから、次のループの実行に入る必要があることを示している。

【0019】しかし、図17のようなCDGに基づいた従来の高位合成方法では、ひとつのループの内部動作の終了を待たずに、次のループの内部動作を実行するようなパイプラインアーキテクチャを合成することはできない。

【0020】但し、従来から行われている他の高位合成方法によれば、入力記述からループをなくすことにより、パイプラインアーキテクチャを合成することができ。そして、このような高位合成方法が、“Proceedings of IEEE ISCAS 87”. (pp. 382-385) においてループワインディング法として提案されている。この高位合成方法は、ループ回数Nが固定（定数）の場合に適用可能である。その手法は、ループの内部動作及びカウンタのインクリメントの動作をN回コピーし（ループの展開）、数2のようなループのない記述に変換し、この記述に対して、従来の高位合成を行なう手法である。

【0021】

【数2】

8

```
i=0;
Adr=i+base; /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef; /*calculation*/
output(Adr,Res); /*output to a port*/
```

```
i=1;
Adr=i+base; /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef; /*calculation*/
output(Adr,Res); /*output to a port*/
```

```
i=2;
Adr=i+base; /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef; /*calculation*/
output(Adr,Res); /*output to a port*/
```

```
...
```

```
i=N-1;
Adr=i+base; /*set address*/
Data=input(Adr); /*input from a port*/
Res=Data*Coef; /*calculation*/
output(Adr,Res); /*output to a port*/
```

【0022】数2の記述に対応するCDGは、図20に示したCDGである。数2の記述は、条件判断のない演算のみの記述であり、コントロールフローグラフは不要となる。図20のデータ依存グラフに基づけば、従来の高位合成方法によっても、パイプライン状の実行を行なうスケジューリングが可能となる。しかし、コピーを行うことでループを展開しなければならないため、ループ回数が定数である場合以外適用できない。

【0023】このように、従来の高位合成方法では、任意の回数繰り返すループを含んだ入力記述から高性能なパイプラインアーキテクチャを合成することができない。そのため、非パイプラインアーキテクチャを合成するか、人手でパイプラインアーキテクチャを用いたRTL記述を作成しなければならない。しかし、前者では高性能化が達成できず、後者では設計期間の長期化、設計者の負担の増大などの問題が生ずる。

【0024】この発明は上記のような問題を解決するためになされたもので、任意の回数繰り返すループを含んだ入力記述から、高性能なパイプライン型のデータバス構成を有するアーキテクチャ及びその制御論理を自動合成することを目的とする。

【0025】

【課題を解決するための手段】第1の発明に係る自動高

位合成方法は、ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程と、計算機を用いて、自動的に、前記ループ記述に対応したコントロールフローグラフ及び前記データ依存グラフを、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程とを備えて構成される。

【0026】第2の発明に係る自動高位合成方法は、第1の発明の自動高位合成方法において、前記変形工程は、前記ループ記述の内部動作を表す前記データ依存グラフの入力変数を要素とする第1の集合と、前記データ依存グラフの出力変数を要素とする第2の集合とを生成する工程と、前記第1の集合と前記第2の集合との積集合を生成する工程と、前記積集合に属する出力変数を出力するノードから、前記データ依存グラフを入力側へトレースして、前記出力変数を出力するために必要な演算を行う第1のノードとそれ以外の演算を行う第2のノードとに分離抽出する工程と、前記新たなデータ依存グラフとして、前記第1のノードからなる第1のデータ依存グラフと前記第2のノードからなる第2のデータ依存グラフとを生成するとともに、前記新たなコントロールフローグラフとして、前記第1及び第2のデータ依存グラフの実行手順を規定するコントロールフローグラフを生成する工程とを含むことを特徴とする。

【0027】第3の発明に係る自動高位合成方法は、第1または第2の発明の自動高位合成方法において、前記計算機に準備した演算器のデータの中から、前記新たなデータ依存グラフ中の全ての前記ノードにそれぞれ一対一に対応する演算器のデータを関連づけるバインディング工程と、前記計算機において、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための前記演算器の制御論理及びデータバス構造を自動生成する生成工程とを備えて構成される。

【0028】第4の発明に係る自動高位合成方法は、第3の発明の自動高位合成方法において、前記生成工程は、前記ループの脱出条件を判定するノードを第1のステップに割り付ける第1のスケジューリング工程と、前記第1のノードに対応する前記演算器について、前記第1のデータ依存グラフの制約に基づいて、第2のステップ以降に順に割り付け、制御ステップの決定を行う第2のスケジューリング工程と、前記第2のノードに対応する前記演算器について、前記第2のデータ依存グラフの制約に基づいて、制御ステップの決定を行う第3のスケジューリング工程と、接続すべき前記演算器の出力と入力の前記制御ステップが異なっているときには、その差異段数と等しい数のパイプラインレジスタを挿入した後に、差異段数がないときにはそのまま通常のリソース接続を行うリソース接続工程とを備えて構成される。

【0029】第5の発明に係る自動高位合成方法は、第3または第4の発明の自動高位合成方法において、前記演算器の遅延値が1クロック周期以上の場合に、前記演算器をラッチで分離されたパイプライン演算器としてモデル化する工程を備えて構成される。

【0030】第6の発明に係る自動高位合成方法は、第3乃至第5の発明の自動高位合成方法において、前記生成工程は、遅延値が1クロック周期以上の前記演算器に対応して一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やす工程を含むことを特徴とする。

【0031】第7の発明に係る自動高位合成方法は、第3乃至第6の発明の自動高位合成方法のいずれかにおいて、前記生成工程は、一つの前記ループの処理を完了するために必要な工程数であるパイプラインピッチを増やすとともに複数の前記ノードにおいて一部の演算器を共有させる工程を含むことを特徴とする。

【0032】第8の発明に係る自動高位合成方法は、第3乃至第7の発明の自動高位合成方法のいずれかにおいて、前記新たなコントロールフローグラフ及びデータ依存グラフによって表される回路の一部のパイプラインレジスタの占有面積と前記パイプラインレジスタによって与えられる値を出力する所定の演算を行う回路の占有面積とを比較する比較工程と、前記比較工程によって、前記所定の演算を行う回路を用いる方が回路面積を小さくできると判断された場合に、前記パイプラインレジスタを前記所定の演算を行う回路に置換する置換工程とを備えて構成される。

【0033】第9の発明に係る自動高位合成方法は、第1乃至第8の発明の自動高位合成方法のいずれかにおいて、前記ループ記述に対応したコントロールフローグラフ及びデータ依存グラフを準備する工程は、前記コントロールフローグラフ及びデータ依存グラフで表すべき回路を表現した動作レベルのハードウェア記述言語による記述を入力する工程を含み、前記変形工程で変形された前記新たなコントロールフローグラフ及びデータ依存グラフを表現した、論理合成ツールが直接処理可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力する工程を備えて構成される。

【0034】第10の発明に係る自動高位合成方法は、第9の発明の自動高位合成方法において、複数の演算によって構成される一つの関数を表す新たな演算器を追加登録する工程と、前記新たな演算器を他の前記演算器と同様に処理するために、前記動作レベルのハードウェア記述言語で特殊なコメントを挿入する工程とを備えて構成される。

【0035】第11の発明に係る自動高位合成方法は、第9の発明の自動高位合成方法において、前記ループ記述で表現されたループのうち複数回分のループを、1回分のループとして表現するループ記述に自動展開する工

程を備えて構成される。

【0036】

【作用】第1の発明の自動高位合成方法におけるループ記述を特定する工程によって、ループ記述に対応するコントロールフローグラフ及びデータ依存グラフを準備する。そして、変形工程において、ループ記述に対応するコントロールフローグラフ及びデータ依存グラフを新たなコントロールフローグラフ及びデータ依存グラフへと変形する。変形工程で得られた新たなコントロールフローグラフ及びデータ依存グラフは、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す。

【0037】第2の発明の自動高位合成方法における入出力変数をそれぞれ構成要素とする第1及び第2の集合を生成する工程と、それらの積集合を生成する工程とによって、前のループの演算結果を次のループの演算結果に反映するための変数を特定することができる。そして、積集合に属する出力変数を出力するノードから、データ依存グラフを入力側へトレースして、出力変数を出力するために必要な演算を行う第1のノードとそれ以外の演算を行う第2のノードとに分離抽出する工程と、新たなデータ依存グラフとして、第1のノードからなる第1のデータ依存グラフと第2のノードからなる第2のデータ依存グラフとを生成するとともに、新たなコントロールフローグラフとして、第1及び第2のデータ依存グラフの実行手順を規定するコントロールフローグラフを生成する工程とによって、パイプライン状に演算を実行可能な第2のデータ依存グラフを生成することができる。

【0038】第3の発明の自動高位合成方法におけるバインディング工程によって新たなデータ依存グラフ中の全てのノードにそれぞれ一対一に対応する演算器のデータを関連づけることができる。そうすることによって、生成工程において、計算機において、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータバス構造を容易に自動生成することができる。

【0039】第4の発明の自動高位合成方法における第1ないし第3のスケジューリング工程によって、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器のスケジューリングができ、リソース接続工程によって、接続すべき演算器の出力と入力との制御ステップが異なっているときに、その差異段数と等しい数のパイプラインレジスタを挿入するとともにリソース接続を行うことでパイプライン状に実行したときにデータが消失しないようにパイプラインレジスタを挿入することができる。

【0040】第5の発明の自動高位合成方法におけるモ

デル化する工程で、演算器の遅延値が1クロック周期以上の場合に、演算器をラッチで分離されたパイプライン演算器としてモデル化され、そのモデル化された演算器のラッチで分離された部分を一つの演算器と同様に扱うことによって、容易に、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータバス構造を容易に自動生成することができる。

【0041】第6の発明の自動高位合成方法における生成工程は、一つのループの処理を完了するために必要な工程数であるパイプラインピッチを増やすことによって、遅延値が1クロック周期以上の演算器に対して増やしたパイプラインピッチを割り当てることができ、遅延値が1クロック周期以上の演算器を用いて、リソースが空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータバス構造を容易に自動生成することができる。

【0042】第7の発明の自動高位合成方法における複数のノードにおいて一部の演算器を共有させる工程は、一つのループの処理を完了するために必要な工程数であるパイプラインピッチを増やすことで、リソースが空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御論理及びデータバス構造を容易に自動生成することができる。

【0043】第8の発明の自動高位合成方法における比較工程によれば、新たなコントロールフローグラフ及びデータ依存グラフによって表される回路の一部のパイプラインレジスタの占有面積とパイプラインレジスタによって与えられる値を出力する所定の演算を行う回路の占有面積とを比較して、占有面積の小さい方がどちらかを判断することができる。そして、置換工程において、比較工程により所定の演算を行う回路を用いる方が回路面積を小さくできると判断された場合に、パイプラインレジスタを所定の演算を行う回路に置換することで、占有面積をより小さくすることができる。

【0044】第9の発明の自動高位合成方法におけるコントロールフローグラフ及びデータ依存グラフで表すべき回路を表現した動作レベルのハードウェア記述言語による記述を入力する工程で、動作レベルのハードウェア記述言語で記述したものを入力すると、レジスタトランスファレベルのハードウェア記述言語による記述を出力する工程で、前記変形工程で変形された前記新たなコントロールフローグラフ及びデータ依存グラフを表現した、論理合成ツールが直接処理可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力することができる。そのため、入力が容易になる。

【0045】第10の発明の自動高位合成方法における新たな演算器を追加登録する工程と特殊なコメントを挿入する工程とにより、複数の演算で構成される一つの関

10

20

30

40

50

13

数を表す新たな演算器を追加登録し、動作レベルのハードウェア記述言語で特殊なコメントを挿入することができ、登録された新たな演算器を他の演算器と同様に処理することが可能になる。

【0046】第11の発明の自動高位合成方法における、複数回数分のループを、1回のループに自動展開し、ループの展開が終了した後に、パイプラインアーキテクチャを可能にする変形工程を行うことで、ループの回数を削減したパイプラインアーキテクチャの構成ができる。

【0047】

【実施例】

実施例1. 以下、この発明の第1実施例を図について説明する。数1の入力記述を解析すると、次のループのデータ依存グラフの演算に影響を与えるのは、カウンタのインクリメント ($i=i+1$) だけであり、他の演算についてはループ内部で完結していることがわかる。図1は、複数のループに渡って行われる処理の概念を示す図である。図1において、10aから10yは変数*i*の値の決定と変数*i*の値の範囲の判断とを行う操作、11aから11xは決定された変数*i*を用いて行う内部操作のうちの非依存部である。パイプライン型のデータバス構成を形成するためには、図1のように非依存部11a~11xの動作をオーバーラップさせパイプライン状に実行させることが必要となる。その様子を図2に示す。図2はパイプライン状に実行させたときの各演算とその演算が行われるステップの関係を示す図である。図2において、12はステップ回数を表示する列、13はノードの符号を示した行である。また、図において、14~16は変数*i*の値が0~2の時の処理が行われることを示す記号、17~19は変数*i*の値が $N-3 \sim N-1$ の時の処理が行われることを示す記号である。例えば、ノードE1及びノードE2は操作10a~10yに属し、ノードE3~E6は操作11a~11xに属するものとする。パイプライン状に処理を実行するノードE3~E6は、ノードE1またはノードE2の出力を用いた処理ができる。

【0048】図3及び図4に、ループ記述に対応したCDGから、パイプライン状の実行が可能なCDGへ変換する処理のフローチャートを示す。コンピュータにおいて、図17のCDGを入力して、パイプライン状の実行が可能なCDGへの変換手順をフローチャートに沿って説明する。

【0049】まず、工程S11において、図17に示した従来の高位合成方法でループ記述から変換されたCDGを入力する。次に、工程S12において、ループの内部動作を表すデータ依存グラフ(図17のブロック3)に外から入ってくる変数の集合Iと外へ出ていく変数の集合Oを生成する。図17の例では、集合I = {1, i, base, Coef}、集合O = {i} となる。

14

【0050】続いて、工程S13において、集合Iと集合Oの交差集合(積集合)Cを求める。図17の例では、交差集合C = {i} となる。次に、工程S14において、ブロック3に対して、交差集合Cに含まれる出力変数*i*から、入力に向かって枝をトレースする。

【0051】工程S15において、先の工程S14でトレースされたノード(演算)の集合Uを求める。図17の例では、集合Uは加算ノードE2となる。工程S16において、集合Uの要素を、次のループに影響を与える演算として抽出する。工程S17において、集合Uに含まれない演算を、ループ間に依存関係のない演算として抽出する。

【0052】工程S18において、次のループに影響を与える演算の集合と、依存関係のない演算の集合について新たなブロックをつくり、それぞれのブロックへ、条件比較ブロック2からコントロール枝を引く。さらに、依存関係のあるブロックについては、ループのためのコントロール・フローの枝を、そのブロックから条件比較ブロックへ引き、コントロールフローグラフを生成する。そして、それぞれのブロック内で、再度データ依存グラフを構成しなおす。工程S19において、変形することにより得られたCDGを出力する。以上のような処理により、図17のCDGを、図5のようなパイプライン状の実行が可能なCDGへ変形できる。

【0053】以上のようにして第1の実施例の高位合成方法は、ループ記述に対応するCDGを、図5のようなパイプライン状の実行が可能な形態のCDGへと変換することができる。図5において、4はカウンタのインクリメント ($i=i+1$) を行うブロック、5は数1のループ依存関係の無い内部動作を行うブロックである。

【0054】図5のCDGは、図17のCDGを、ループ間にまたがって依存関係のある演算のノードE2を含むブロック4と依存関係のない演算のノードE3~E6のみからなるブロック5に分けたものである。図5に示したCDGは、非依存部の演算については、前のループの演算終了を持つことなく次のループの演算実行が可能であることを意味している。

【0055】次に、図5に示したCDGを用いて、パイプラインアーキテクチャの合成についての説明を続ける。

【0056】図6及び図7は、パイプラインアーキテクチャ合成のための処理を行なうフローチャートである。図5のCDGを入力例として、このフローチャートを具体的に説明する。

【0057】まず、工程S21において、図5に示したような変形されたCDGをコンピュータが入力する。次に、工程S22において、その演算を実行可能な演算器をCDGの個々の演算に対して1つ割り付ける。この時、割り付けられる各演算器の最大遅延時間は、1クロック周期以内である必要がある。また、同種の演算器を

割り付ける場合、1つの演算器を共有せず、新たな演算器を用意し、それを割り付ける。この演算器の割り付け処理は、リソースバインディングと呼ばれている。 *

*【0058】
【表4】

演算器の種類	最大遅延 (nSec)	コスト (面積)	可能な演算
ADD1	30	100	+
ADD2	60	50	+
MLP1	45	1000	*
MLP2	90	500	*
LSS	20	20	<
INP	45	10	input
OUT1	45	50	output
OUT2	90	10	output

【0059】表4に示すような演算器が利用できるものとし、それらの演算器のデータがコンピュータに準備されているものとする。クロック周期を50nSecとした時の、図5のCDGに対するリソースバインディングの一例は表1で示される。表4を分析すると、演算器ADD2、MLP2、OUT2の演算器は、遅延が1クロック周期(50nSec)より長いので使用できない。また、2つの加算に対しては、異なった演算器ADD1-1、ADD1-2を割り付ける。

【0060】工程S23～工程S25において、これらの演算器が、どの制御ステップにて、どの演算を実行するかを決定する。この制御ステップを決定する処理は、スケジューリングと呼ばれている。

【0061】工程S23において、ループの脱出条件を判定する演算は、第0ステップにて実行させるように割り付ける。ループにまたがって依存関係のある全ての演算は、第1ステップにて実行させる。もし、第1ステップにて実行できなければ、高位合成不能である。つまり、第1のステップに依存関係の無いノードが割り付けられると処理ができなくなるからである。

【0062】最後に、次のループへ影響しない演算に対して、制御ステップを決定する。この問題は、必要なレジスタ数を評価関数としたスケジューリング問題となる。スケジューリング問題の解法については、従来の高位合成と同様の手法を用いる。Daniel Gajski, Nikil Dutt, Allen Wu, Steve Lin著, "High-Level Synthesis", (1992), KLUWER ACADEMIC PUBLISHERSに様々な解法が紹介されている。演算をできるだけ早いステップで動作させるスケジューリング法である、ASAP法と呼ばれる解法を用いて、図5のCDGと表1のリソースバインディングに対して、スケジューリングした結果は表2に示したものと同様である。

※【0063】工程S26において、使用された演算器を直接、または、レジスタやセクタを介して接続し、データバス構造を生成する。この処理はリソース接続と呼ばれている。リソース接続は、次のようなルールに従って行なわれる。データ依存グラフにおいて、データの授受を行う関係にある2つのノードに対応する演算器の制御ステップが前後の関係にあるなら、言い換えると、それらの演算器の入力と出力とを同じ制御ステップで行うことができるなら、演算器同士を直接接続する。もし、動作すべき制御ステップが離れていれば、制御ステップの差分の段数のレジスタ(パイプラインレジスタと呼ぶ)を介して接続する。

【0064】また、次のループに影響を与えるオペレーションには、ループ内動作に対応する演算器の出力端子にレジスタを挿入し、次の動作の入力へ接続する。但し、同一の演算器やレジスタの入力端子に、複数の演算器やレジスタを接続する必要がある場合には、セクタを生成し、セクタを介して接続する。図8に、表2のスケジューリング結果に対して、リソース接続を行なった例を示す。図8において、r1～r11はレジスタ、20は第2の入力端子に接続されたレジスタr2に記憶されている値0と第1の入力端子に入力される値とを切り替えて出力するセクタ、21はセクタ20が出力した値を保持しているレジスタr4の値とレジスタr1が保持しているループの繰り返し回数値とを比較して結果に応じて信号S1を出力する比較器、22はセクタ20等の各演算器を制御するための制御論理回路ブロック、23はレジスタr3に記憶されている値1とレジスタr4に記憶されている値とを加算してその結果をセクタ20の第1の入力端子に与える加算器、24はレジスタr4に記憶されている値とレジスタr5に記憶されている値とを加算してその結果をレジスタr6に出力

※50

17

する加算器、25はレジスタ6の値に応じてデータを
入力してレジスタ7に出力する入力装置、26はレジ
スタ7に記憶されているデータとレジスタ8に記憶
されている係数とを掛け合わせてその結果をレジスタ
11に出力する乗算器、27はレジスタ10とレジス
タ11との値に応じた出力を行う出力装置である。な
お、レジスタ6からレジスタ9へ、またレジスタ9
からレジスタ10へステップ毎に順に記憶されてい
るデータが送られる。以上で、パイプライン動作部のデ
ータパス構造が合成できた。

【0065】続いて、工程S27において、制御論理回
路ブロック22が出力するための制御論理を生成する。
制御論理は、LSIの動作を制御するための、状態遷移
として表現され、適切な状態での演算器やレジスタやセ
レクタへの動作信号の発生や、条件に応じた次状態を規
定している。制御論理の生成法を説明する。スケジュー
リングで求められた制御ステップの個数を(N+1)個
とする。この時、(2×N+1)個の状態を用意する。
(2×N+1)個の状態は、(1)初期値設定、(2)初期
判断、(3)パイプラインの導入部、(4)フル・パイプ
ライン動作部、(5)パイプライン導出部のブロックに分割
できる。

【0066】ステート0は、初期値設定ブロックであ
る。ここでは、カウンタに対応するレジスタに初期値を
設定する。設定に必要なセレクタとレジスタへの制御信
号を発生する。次状態は、ステート1とする。

【0067】ステート1は、初期判断ブロックである。
これは、ループを一度も実行しないケースを想定してい
る。この状態においては、ループの脱出条件判定を行な*

18

*う演算器が動作する。この演算器に対する制御信号が必
要ならその制御信号を発生する。図8の例では、不要で
あるため、制御信号は生成しない。次状態は、ループの
条件判定の演算器の出力が真(1)ならステート2とし、
偽(0)なら、終状態(END)とする。

【0068】ステート2からステートNまでは、パイ
プライン導入部である。ステート2からステートNの中の
任意のステートiでは、スケジューリング結果のステッ
プ0～i-1までの制御ステップに関わる演算器、レジ
スタ、セレクタに対して、必要な制御信号を生成する。
次状態はループの条件判定の演算器の出力が真(1)なら
ステートi+1とし、偽(0)なら、ステートN+2とす
る。

【0069】ステートN+1は、フル・パイプライン動
作部である。この状態では、全ての演算器を動作させ
る。次状態は、ループの条件判定の演算器の出力が真
(1)ならステートN+1(同じ状態)とし、偽(0)な
ら、ステートN+2とする。

【0070】ステートN+2からステート2Nまでは、
パイプライン導出部である。ステートN+2からステ
ート2Nまでの任意のステートiでは、スケジューリング
結果のステップi-N～Nまでの制御ステップに関わる
演算器、レジスタ、セレクタに対して、必要な制御信号
を生成する。次状態はステート2N以外の状態では、ス
テートi+1とし、ステート2Nの状態では、終状態
(END)とする。

【0071】

【表5】

現状態	演算器名			レジスタ					
	INP	OUT	セレクタ	i	Adr	Data	Adr (1)	Res	Adr (2)
ステート0			select 2	WRITE					
ステート1									
ステート2			select 1	WRITE	WRITE				
ステート3	active		select 1	WRITE	WRITE	WRITE	WRITE		
ステート4	active		select 1	WRITE	WRITE	WRITE	WRITE	WRITE	WRITE
ステート5	active	active	select 1	WRITE	WRITE	WRITE	WRITE	WRITE	WRITE
ステート6	active	active				WRITE	WRITE	WRITE	WRITE
ステート7		active						WRITE	WRITE
ステート8		active							

【0072】

※ ※【表6】

現状態	次状態	
ステート0	ステート1	初期値設定
ステート1	ステート2 (S1=1) END (S1=0)	初期値判断
ステート2	ステート3 (S1=1) ステート6 (S1=0)	パイプライン 導入部
ステート3	ステート4 (S1=1) ステート6 (S1=0)	
ステート4	ステート5 (S1=1) ステート6 (S1=0)	
ステート5	ステート5 (S1=1) ステート6 (S1=0)	
ステート6	ステート7	フル・パイプライン
ステート7	ステート8	
ステート8	END	

【0073】表5、表6に、上記の高位合成にて得られる制御論理を示す。図8に示した構成と、表5及び表6に示した制御論理によって、数1の入力記述を実現するパイプラインアーキテクチャが得られている。

【0074】最後に、工程S28において、得られたアーキテクチャを出力する。以上の処理により、パイプラインアーキテクチャを合成することができる。また、リソースコストの増加を抑えて処理の高速化を実現することができる。

【0075】実施例2。次に、この発明の第2実施例による高位合成方法について説明する。上記の高位合成方法では、各演算器の最大遅延時間が、1クロック周期以内である必要がある。しかし、このような制限があると、使用可能な演算器が限定されてしまい、与えられる条件によっては、このような制限のためにアーキテクチャの合成が不可能な場合が出てくる。

【0076】まず、このような問題点を解消するための概念について説明する。図9に示すように、出力の入力に対して遅延時間Delayの組合せ回路のみからなる演算器30があり、遅延時間Delay>1クロック周期Clock-Periodとする。この演算器を用いた場合、一度データを入力すると、出力結果が得られるまでの、 $(\text{int}(\text{Delay}/\text{Clock-Period}) + 1)$ クロック周期の間、この演算器を使用することができない。但し、 $\text{int}(x)$ は、実数 x を越えない最大の整数を与える関数であり、 $(\text{Delay}/\text{Clock-Period})$ が整数の時は例外的に $(\text{Delay}/\text{Clock-Period})$ クロック周期の間使用できない。パイプラインアーキテクチャでは、データをクロック毎に演算器に投入し、処理することが必要である。したがって、このような演算器は、パイプラインアーキテクチャでは使用できない。このような演算器を使用可能とする手法に、パイプラインラッチの挿入法がある。まず、各段の実行が1クロック周期内に終るように演算器をN段のブ* 50

* ロックに分割する。分割されたブロック間の接続にラッチを挿入する。パイプラインラッチの挿入の例を、図10に示す。図10において、組合せ回路31〜36で組合せ回路30と同等の機能を有している。そして、組合せ回路31、33の出力はそれぞれラッチ32、34に保持される。このようにラッチが挿入された演算器には、データをクロック毎に投入することができる。但し、演算器の計算結果が得られるのは、Nクロック周期後である。遅延時間が1クロック周期以上の演算器に対しては、このようにして、パイプラインラッチを挿入した演算器へ設計者が変換し、あるいは自動変換し、その後上記と同様の高位合成方法を用いて、パイプラインアーキテクチャの合成を行なう。

【0077】実施例3。しかし、上記のような回路を分割する方法は、組合せ回路のみからなる演算器の場合に適用可能である。ところが、例えば、メモリや外部ポートとのインターフェース回路等では、演算器は組合せ回路のみで構成されず、上記の方法では、パイプラインアーキテクチャを合成することはできない。

【0078】1クロック周期以上の遅延を持つ演算器が組合せ回路以外の回路で構成されている場合も、パイプラインアーキテクチャを合成可能とするこの発明の第3実施例による高位合成方法について説明する。

【0079】表4の利用可能な演算器のテーブルにおいて、OUT1という種類の演算器がなかったとする。クロック周期を50nSecとすれば、出力に対応する演算器で、遅延が1クロック周期以内のものは存在しない。したがって、2クロック周期必要な演算器OUT2を用いて、パイプラインアーキテクチャを合成することを考える。

【0080】

【表7】

21

22

演算器名	LSS	ADD1-1	ADD1-2	INP	MLP1	OUT
ステップ0	E1					
ステップ1		E2	E3			
ステップ2				E4		
ステップ3					E5	
ステップ4						E6
ステップ5						E6

【0081】OUT2の動作には2クロック必要であるため、表2のスケジューリングが、表7のように変化する。図6及び図7のフローチャートに示した手順を含む高位合成方法では、各演算器は一つの制御ステップでのみ動作する必要がある。そのため、表7のようなスケジューリング結果に対しては、正しい制御論理を生成することができない。

【0082】そこで、制御論理の生成方式を以下のように変更する。スケジューリングで求められた制御ステップの個数を $(N+1)$ 個とする。また、遅延時間が最長の演算器に必要なクロック周期を m 周期とする。さらに、 $L=ra(N/m)$ とする。ただし、 $ra(a)$ は、実数 a の少数点以下を切り上げた整数を意味する。

【0083】この時、 $(2mL-m+2)$ の状態を用意する。これらの状態は、(1)初期値設定、(2)初期判断、(3)パイプラインの導入部、(4)フル・パイプライン動作部、(5)パイプライン導出部のブロックに分割できる。

【0084】初期値設定と初期判断のブロックについては、図7の工程S27と同様に、ステート0は、初期値設定ブロックである。ここでは、カウンタに対応するレジスタに初期値を設定する。設定に必要なセレクトとレジスタへの制御信号を発生する。次状態は、ステート1とする。

【0085】ステート1は、初期判断ブロックである。これは、ループを一度も実行しないケースを想定している。この状態においては、ループの脱出条件判定を行なう演算器が動作する。これらの演算器に対する制御信号が必要ならその制御信号を発生する。図8の例では、不要であるため、制御信号は生成しない。次状態は、ループの条件判定の演算器の出力が真(1)ならステート2とし、偽(0)なら、終状態(END)とする。

【0086】ステート2からステート $mL-m+1$ までは、パイプライン導入部である。ステート2からステート*

* $mL-m+1$ の中の任意のステート $i+1$ ($i=1, 2, \dots, mL-m$) に対する制御論理は以下のように定まる。 $t=i \bmod m$ とする。このとき、スケジューリング結果のステップ $t, t+m, t+2m, \dots, i$ の制御ステップに関わる演算器、レジスタ、セレクトに対して、必要な制御信号を生成する。次状態については、 t が0の時とそれ以外の時で異なる。 t が0の時はループの条件判定の演算器の出力が真(1)ならステート $i+1$ とし、偽(0)なら、ステート $mL+2$ とする。 t が0以外の時は、ループの条件判定の演算器の出力とは無関係にステート $i+1$ とする。

【0087】ステート $mL-m+2$ からステート $mL+1$ は、フル・パイプライン動作部である。ステート $mL-m+2+i$ ($i=0, 1, \dots, m-1$) では、 $s \bmod m \equiv i$ となるようなステップ s の制御ステップに関わる演算器、レジスタ、セレクトに対して、必要な制御信号を生成する。次状態については、 i が $m-1$ の時は、ループの条件判定の演算器の出力が真(1)ならステート $mL-m+2$ とし、偽(0)なら、ステート $mL+2$ とする。 i が $m-1$ 以外の時は、常にステート $mL-m+2+i+1$ とする。

【0088】ステート $mL+2$ からステート $2mL-m+1$ までは、パイプライン導出部である。ステート $mL+i+1$ ($i=1, 2, \dots, mL-m$) に対する制御論理は以下のように定まる。 $t=i \bmod m$ とする。このときスケジューリング結果のステップ $i, i+m, i+2m, \dots, mL-m+2+t$ の制御ステップに関わる演算器、レジスタ、セレクトに対して、必要な制御信号を生成する。次状態は、ステート $2mL-m+1$ 以外の状態では、ステート $mL+i+2$ とし、ステート $2mL-m+1$ の状態では、終状態(END)とする。

【0089】

【表8】

現状態	演算器名			レジスタ					
	INP	OUT	セレクト1	i	Adr	Data	Adr (1)	Res	Adr (2)
ステート0			select 2	WRITE					
ステート1									
ステート2			select 1	WRITE	WRITE				
ステート3	active					WRITE	WRITE		
ステート4			select 1	WRITE	WRITE			WRITE	WRITE
ステート5	active	active				WRITE	WRITE		
ステート6		active	select 1	WRITE	WRITE			WRITE	WRITE
ステート7	active	active				WRITE	WRITE		
ステート8		active						WRITE	WRITE
ステート9		active							
ステート10		active							
ステート11									

【0090】

* * 【表9】

現状態	次状態	
ステート0	ステート1	初期値設定
ステート1	ステート2 (S1=1) END (S1=0)	初期値判断
ステート2	ステート3	パイプライン 導入部
ステート3	ステート4 (S1=1) ステート8 (S1=0)	
ステート4	ステート5	
ステート5	ステート6 (S1=1) ステート8 (S1=0)	
ステート6	ステート7	フル・パイプライン
ステート7	ステート8 (S1=1) ステート8 (S1=0)	
ステート8	ステート9	パイプライン 導出部
ステート9	ステート10	
ステート10	ステート11	
ステート11	END	

【0091】表8、表9に表7のスケジューリングに対して、上記の手順を適用したとき得られる制御論理を示す。この例では、 $N=5$ 、 $m=2$ 、 $L=3$ となる。従って、ステップ2～5がパイプライン導入、ステップ6～7がフル・パイプライン部、ステップ8～11がパイプ

※ライン導出部となる。

【0092】上記の方法では、フル・パイプライン部において、 m 個の状態でもってループを構成している。このループを構成する状態数をパイプラインピッチと呼ぶ。図6、図7を用いて説明した方法では、パイプライ

25

ンピッチが1であったため、フル・パイプラインの状態では、スルーアットとしては、1クロックあたり1ループの動作が処理されていた。一方、この方法によれば、mクロックあたり1ループの動作が処理されている。従って、スルーアットが低下したパイプラインアーキテクチャが合成されることとなる。

【0093】実施例4. 次に、この発明の第4実施例による高位合成方法について説明する。第1実施例による高位合成方法では、グラフ中の各演算器に一对一に対応する演算器を用意する必要がある。これが不可能な場合*10

26

*には適用できない。そこで、一部の演算器を共有しても、パイプラインピッチを増やすことにより、第1実施例の高位合成方法を適用可能とする方式に関するものである。

【0094】ここでは、2つの加算(E2, E3)に対して1つの加算器が割り付けることを考える。そのため、図5のCDGに対して、表10のようなリソースバインディングを行ったとする。

【0095】

【表10】

ノード	演算	バインドした演算器の種類	演算器名
E1	<	LSS	LSS
E2	+	ADD1	ADD1
E3	+	ADD1	ADD1
E4	input	INP	INP
E5	*	MLP1	MLP1
E6	output	OUT	OUT

【0096】表11に、このようなリソースバインディングに対するスケジューリングの例を示す。 ※【0097】

※ 【表11】

演算器名	LSS	ADD1	INP	MLP1	OUT
ステップ0	E1				
ステップ1		E3			
ステップ2		E2	E4		
ステップ3				E5	
ステップ4					E6

【0098】1つの演算器を、複数の演算にて共有するために、制御ステップをずらして動作するようにスケジューリングをする。第1実施例の高位合成方法では、各演算器は一つの制御ステップでのみ動作する必要がある。このようなスケジューリング結果に対しては、正しい制御論理を生成することができない。

【0099】このような場合においても、スケジューリング方式を変更し、さらにこの発明の第3実施例による高位合成方法の制御論理の生成方式を用いることにより、パイプラインアーキテクチャが合成可能となる。

【0100】スケジューリング方式の変更点は、スケジューリングに次のような制限を設けることである。一つ★

★の演算器に割り付けられた演算の最大数をmとする。また、ある演算器が動作する制御ステップを S_1, S_2, \dots, S_k ($k \leq m$)とする。このとき、 $S_i \bmod m$ と、 $S_j \bmod m$ とは等しくないという制限を設けて、スケジューリングを行なう。ただし、 $i, j = 1, 2, \dots, k$ として、 i と j は等しくないものとする。

【0101】そして、共有する加算器に入力されるデータの切り替えを行うセレクトを新たに設ける。このセレクトは、制御論理によって、ステップ1とステップ2でその切り替えを行う。

【0102】

【表12】

現状態	演算器名				レジスタ					
	INP	OUT	セクタ1	セクタ2	i	Adr	Data	Adr(1)	Res	Adr(2)
ステート0			select 2		WRITE					
ステート1										
ステート2				select 1	WRITE					
ステート3			select 1	select 2	WRITE					
ステート4	active			select 1		WRITE	WRITE	WRITE		
ステート5			select 1	select 2	WRITE				WRITE	WRITE
ステート6	active	active		select 1		WRITE	WRITE	WRITE		
ステート7			select 1	select 2	WRITE				WRITE	WRITE
ステート8	active	active					WRITE	WRITE		
ステート9									WRITE	WRITE
ステート10		active								
ステート11										

【0103】

* * 【表13】

現状態	次状態	
ステート0	ステート1	初期値設定
ステート1	ステート2 (S1=1) END (S1=0)	初期値判断
ステート2	ステート3	パイプライン 導入部
ステート3	ステート4 (S1=1) ステート8 (S1=0)	
ステート4	ステート5	
ステート5	ステート6 (S1=1) ステート8 (S1=0)	
ステート6	ステート7	
ステート7	ステート8 (S1=1) ステート8 (S1=0)	フル・パイプライン
ステート8	ステート9	パイプライン 導出部
ステート9	ステート10	
ステート10	ステート11	
ステート11	END	

【0104】表12及び表13並びに図11に、表10のリソースバインディングと表11のスケジューリングに対して、リソース接続と制御論理の生成を行ない、パイプラインアーキテクチャ合成した結果を示す。図11において、セクタ28が新たに付け加えられたセクタ※50

※タであり、それによって、加算器23を共有化している。

【0105】実施例5。次に、この発明の第5実施例の高位合成方法について説明する。第1実施例の高位合成方法では、演算器間に制御ステップの差異分の段数のパ

パイラインレジスタが生成される。しかし、差異が大きい場合、パイラインレジスタを用いてデータを保持するより、必要な時点で再計算した方が良い場合がある。このような場合に、データバス構造の構成に変更を加える工程を追加する。

【0106】図8のデータバス構造には、 Adr 、 $Adr(1)$ 、 $Adr(2)$ と続くパイラインレジスタが存在する。この例では、3サイクル前の $i+base$ の演算結果の値を保持するために、パイラインレジスタが作られている。しかし、 i は1サイクルに1インクリメントされているので、3サイクル前の $i+base$ の値は、現在の加算器の出力から、3を減じることによって求めることもできる。もし、2段のパイラインレジスタ(1段目のパイラインレジスタは、消去できない。)を設けるよりも、1つの演算器を導入して値を再計算した方がLSIの面積(コスト)が小さくなる場合、パイラインレジスタを設けずに減算器を新設する。コンピュータによってコスト計算を行いながら高位合成を行うことは、従来から実施されている。

【0107】図12に、図8のデータバス構造のパイラインレジスタを再計算モジュールで置換した例を示す。図12において、29が減算器である。減算器29によって、レジスタ6に記憶されている値からレジスタ13に記憶されている値を引くことによって、出力装置27に対して、レジスタ6が、3ステップ前に保持していた値を与えることができる。

【0108】なお、ここでは、ステップ毎に1ずつ増加する値を保持する場合について示したが、ステップ毎に値が減少する場合であってもよく、また、他の規則によって値が変わる場合であっても同様の効果を奏する。

【0109】次に、この発明の高位合成方法の全体の流れについて説明する。高位合成方法は、ハードウェア記述言語を用いた、動作レベルの記述を入力し、論理合成ツールが直接読み込み可能なレジスタトランスファレベルのハードウェア記述言語による記述を出力する。図13は、この発明の高位合成方法の手順を示すフローチャートである。

【0110】工程S31において、動作レベルのハードウェア記述言語で表現された動作記述を入力し、オペレーション系列へと変換する。工程S32において、オペレーション系列からCDGを生成する。工程S33において、第1実施例と同様に、パイライン状の実行が必要な記述部に対して、パイライン状の実行が可能な形態へと、CDGを変形する。

【0111】工程S34において、変形されたCDGに基づき、第1実施例ないし第5実施例の工程を適用して、パイラインアーキテクチャを合成する。工程S35において、パイライン状の実行が不要な記述部に対して、従来方式にて、アーキテクチャを合成する。この時、工程S34で得られたアーキテクチャ中の演算器を

共有してもよい。最後に、工程S36において、合成されたアーキテクチャを、論理合成ツールが直接読み込むことのできる、レジスタ・トランスファ・レベルのハードウェア記述言語による記述として出力する。

【0112】実施例6. 次に、この発明の第6実施例による高位合成方法について説明する。ループ記述を、コントロールフローグラフと、データ依存グラフに変換する時、通常は、一つの演算子に対して、データ依存グラフ上の1つのノードが生成される。しかし、複雑な演算を実行するゲート回路をユーザが設計し、それに対応する演算子をユーザが追加したい場合がありうる。第6実施例の高位合成方法は、これを容易に可能とする手法に関わるものである。

【0113】ループ動作を含むハードウェアの動作を記述するのに適当な言語として、VHDL、Verilog-HDL等のハードウェア記述言語が考えられる。これらの言語には、複数の入力データを与え、1つの出力を返す関数(function)文がある。

【0114】通常は、この関数文の内部動作まで解析し、演算子レベルにまで分解してCDGを生成する。図14は、Verilog-HDL記述からデータ依存グラフへの通常の変換を示す図である。Verilog-HDL記述40は、データ依存グラフ50に変換される際、一塊の関数50aも演算子のレベルで展開して作成されている。

【0115】しかし、変換ツールへの特殊な指示子をコメント文として挿入するようにすることで、この関数を新しいユーザ定義演算子とみなすようにする。

【0116】これにより、容易に新しいユーザ定義演算子を増やすことができる。図15に、Verilog-HDLでのユーザ定義演算子の例を示す。まず、func60というユーザ定義演算子をコンピュータにあらかじめ登録しておく。そして、Verilog-HDL記述45を作成する際に、`/*UserDefinedFunction*/`という、ユーザ定義演算子の使用を宣言する文を挿入する。そうすることで、データ依存グラフ作成の際も、funcという関数を一つのノードに割り付けることができる。funcという関数に対応する回路は、予めコンピュータに登録されているため、設計者が意図した結果が得られ、性能の良いLSIの設計が可能になる。

【0117】実施例7. 次に、この発明の第7実施例による高位合成方法について説明する。第7実施例の高位合成方法は、入力記述を自動修正し、ループ回数を削減することで、処理速度とリソース数のトレードオフを達成する手法に関するものである。

【0118】合成されたパイラインアーキテクチャは、ループ部の動作に $N+\alpha$ (N はループ回数、 α は定数)サイクル必要とする。従って処理の高速化を図るためには、ループ回数を削減すればよい。

【0119】ループ回数を削減する単純な方法は、数回

31

分のループを1回分に展開することである。例えば、第1実施例の高位合成方法において、2回分のループの記述を1回分に展開する工程をさらに追加することで、ループは、 $N/2 + \beta$ (N はループ回数、 β は定数) サイクルとなり、処理速度は、約2倍となる。数3に展開前の記述を、数4に展開後の記述を示す。

【0120】

【数3】

```
for(i=0;i<N;i=i+1)begin
  Adr=i+base; /*set address*/
  Data=input(Adr); /*input from a port*/
  Res=Data*Coef; /*calculation*/
  output(Adr,Res); /*output to a port*/
end
```

【0121】

【数4】

```
for(i=0;i<N;i=i+2)begin
  i tmp=i;
  Adr=i tmp+base; /*set address*/
  Data=input(Adr); /*input from a port*/
  Res=Data*Coef; /*calculation*/
  output(Adr,Res); /*output to a port*/

  i tmp=i+1;
  Adr=i tmp+base; /*set address*/
  Data=input(Adr); /*input from a port*/
  Res=Data*Coef; /*calculation*/
  output(Adr,Res); /*output to a port*/
end
```

【0122】なお、上記各実施例は、それぞれ組み合わせて用いることもでき、その場合にも、それぞれの固有の効果を失うことはない。

【0123】

【発明の効果】以上のように、請求項1記載の発明の自動高位合成方法によれば、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程を備えているので、新たなコントロールフローグラフ及びデータ依存グラフを用いて、並列度の高いパイプラインアーキテクチャを自動的に合成できるという効果がある。

【0124】請求項2記載の発明の自動高位合成方法によれば、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフへと変形する変形工程を容易に実現することができるという効果がある。

【0125】請求項3記載の発明の自動高位合成方法によれば、前のループの動作完了を待たずに、次のループの動作が実行可能な形態を示す新たなコントロールフローグラフ及びデータ依存グラフを用いて、演算器が空き

32

次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための演算器の制御理論及びデータバス構造を容易に得ることができるという効果がある。

【0126】請求項4記載の発明の自動高位合成方法によれば、計算機において、演算器が空き次第、次のループのデータを投入することにより、ループ動作をパイプライン状に実行するための前記演算器の制御理論及びデータバス構造を自動生成する生成工程を容易に実現できるという効果がある。

【0127】請求項5記載の発明の自動高位合成方法によれば、演算器をラッチで分離されたパイプライン演算器としてモデル化することによって、1クロック周期以上の遅延を有する演算器をパイプラインアーキテクチャにおいて使用できるようにすることができるという効果がある。

【0128】請求項6記載の発明の自動高位合成方法によれば、パイプラインピッチを増やすことによって、1クロック周期以上の遅延を有する演算器をパイプラインアーキテクチャにおいて使用できるようにすることができるという効果がある。

【0129】請求項7記載の発明の自動高位合成方法によれば、一部の演算器を共有にすることによって、LSIの規模を縮小することができるという効果がある。

【0130】請求項8記載の発明の自動高位合成方法によれば、パイプラインレジスタに換えて所定の演算を行う回路を用いて、LSIのサイズを縮小できるという効果がある。

【0131】請求項9記載の発明の自動高位合成方法によれば、入力を動作レベルのハードウェア記述言語で行って、レジスタトランスファレベルのハードウェア記述言語で記述されたパイプラインアーキテクチャを得ることができるという効果がある。

【0132】請求項10記載の発明の自動高位合成方法によれば、ユーザが新たに演算器を登録することができ、より性能の良いパイプラインアーキテクチャを構成することができるという効果がある。

【0133】請求項11記載の発明の自動高位合成方法によれば、展開したループを用いてパイプラインアーキテクチャを生成することができ、ループの回数を減らして処理速度の速いLSIを得ることができるという効果がある。

【図面の簡単な説明】

【図1】 ループの実行状態を示す概念図である。

【図2】 ループの非依存部の動作をパイプライン状に実行する各ノードの動作を示す概念図である。

【図3】 コントロールフローグラフとデータ依存グラフの変形を行なう手順を示すフローチャートである。

【図4】 コントロールフローグラフとデータ依存グラフの変形を行なう手順を示すフローチャートである。

【図5】 バイブライン状の実行が可能な形態のコントロールフローグラフとデータ依存グラフである。

【図6】 バイブラインアーキテクチャを合成する手続のフローチャートである。

【図7】 バイブラインアーキテクチャを合成する手続のフローチャートである。

【図8】 バイブラインアーキテクチャ合成用のリソース接続結果を示すブロック図である。

【図9】 1クロック周期以上の遅延時間をもつ演算器を示す概念図である。

【図10】 1クロック周期以上の遅延時間をもつバイブライン演算器を示す概念図である。

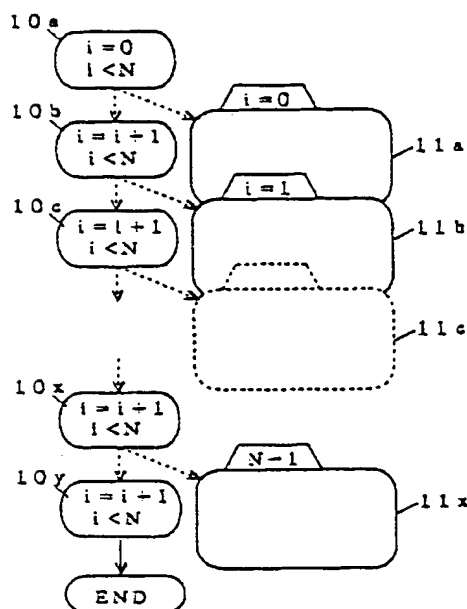
【図11】 演算器を共有化した場合のバイブラインアーキテクチャを示すブロック図である。

【図12】 バイブラインレジスタの再計算モジュールでの置換を示すブロック図である。

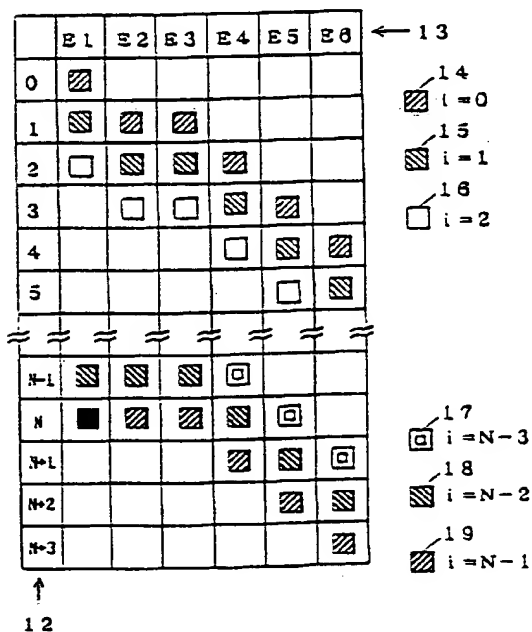
【図13】 バイブラインアーキテクチャを合成する高位合成方法全体の流れを示すフローチャートである。

【図14】 関数を構成する演算子の記述の変換を説明

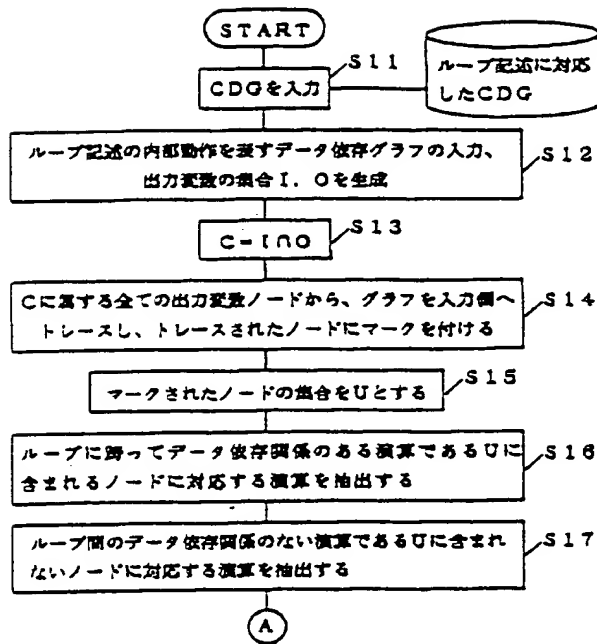
【図1】



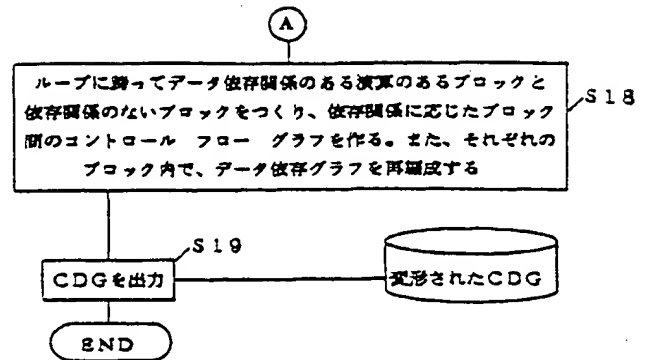
【図2】



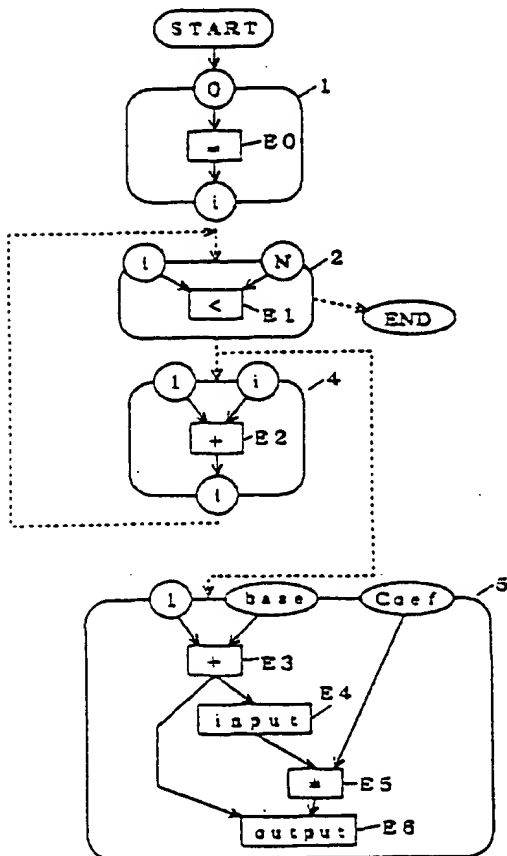
【図3】



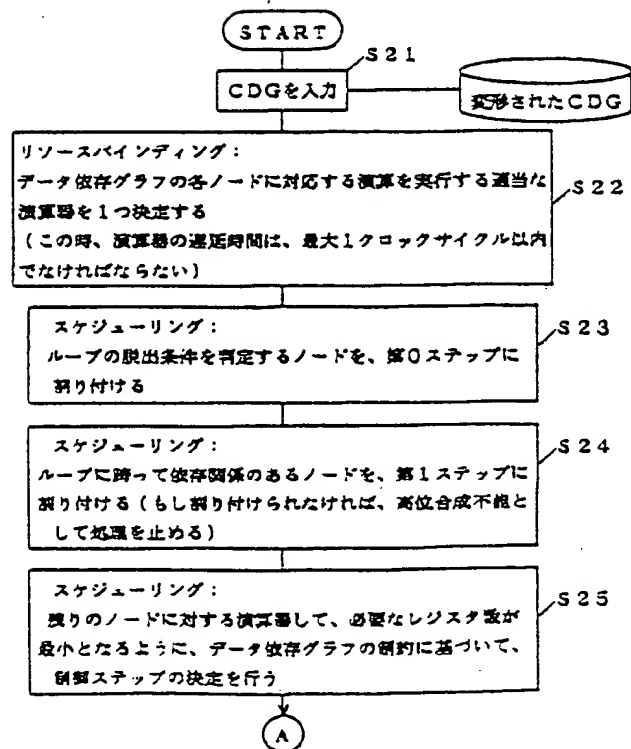
【図4】



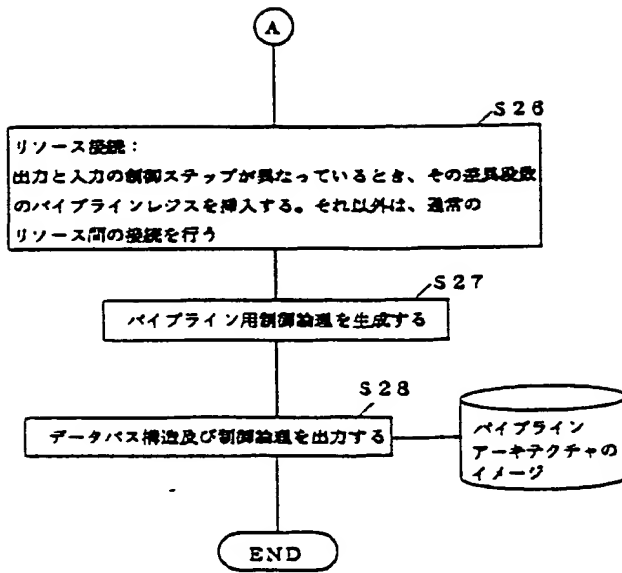
【図5】



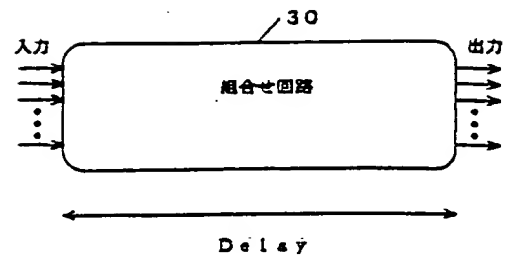
【図6】



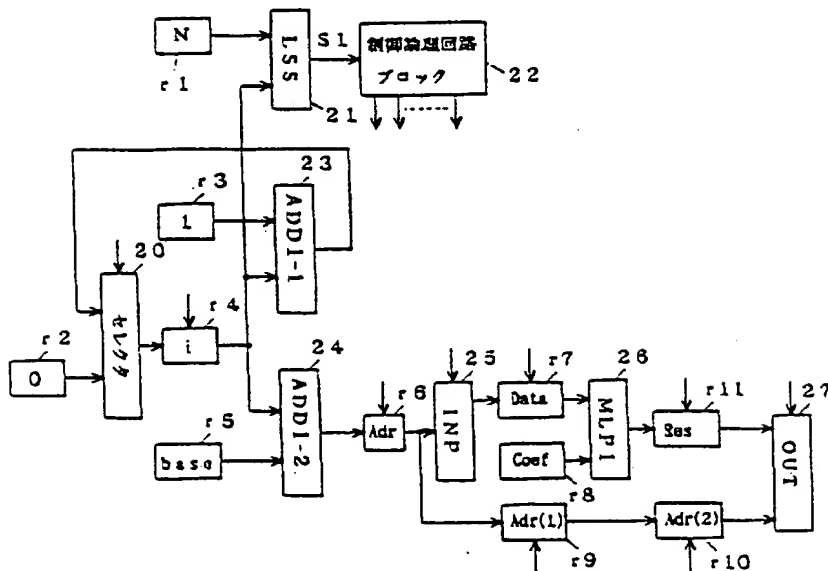
【図7】



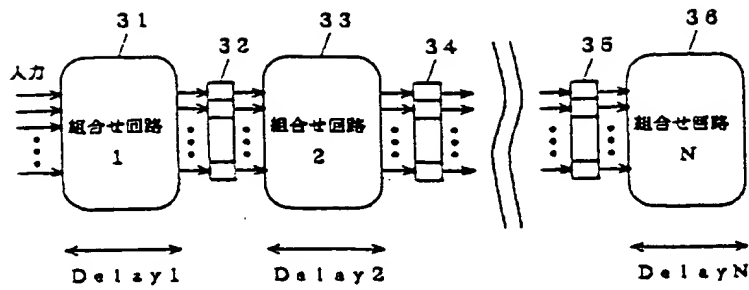
【図9】



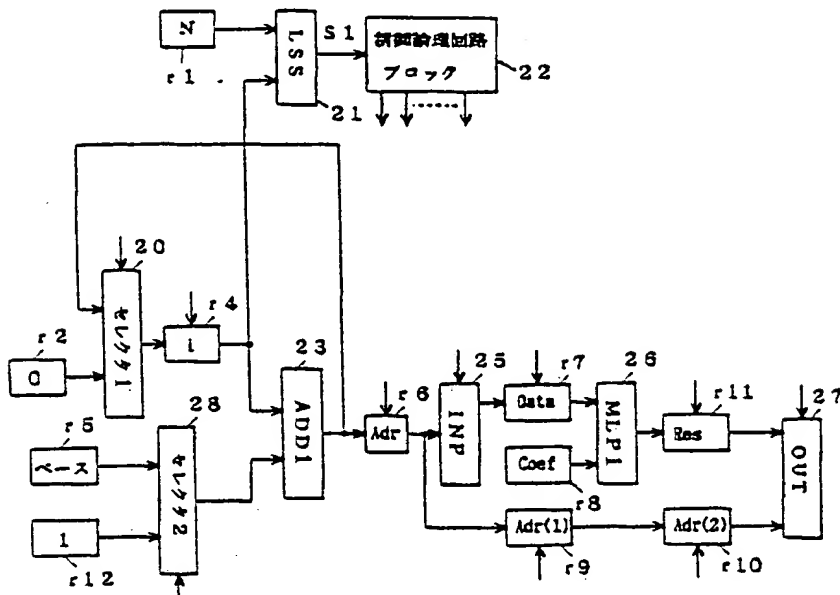
【図8】



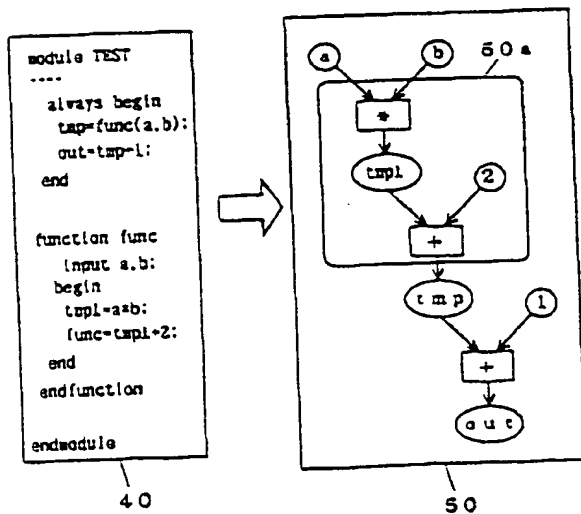
【図10】



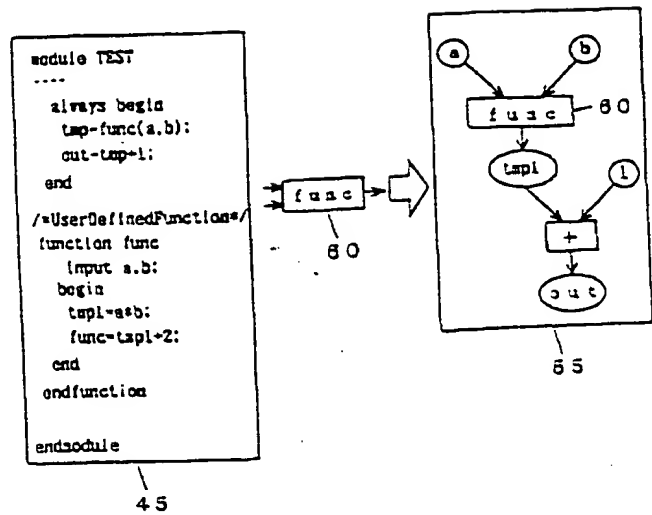
【図11】



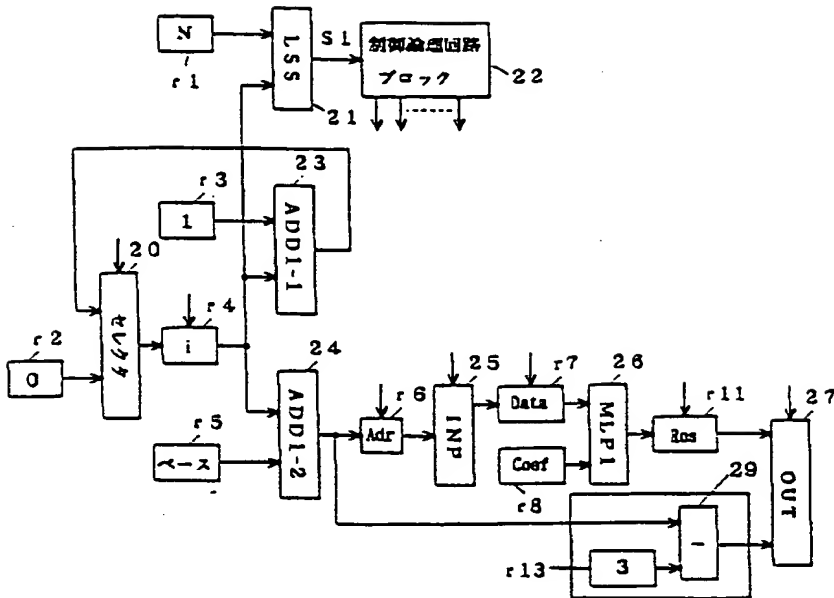
【図14】



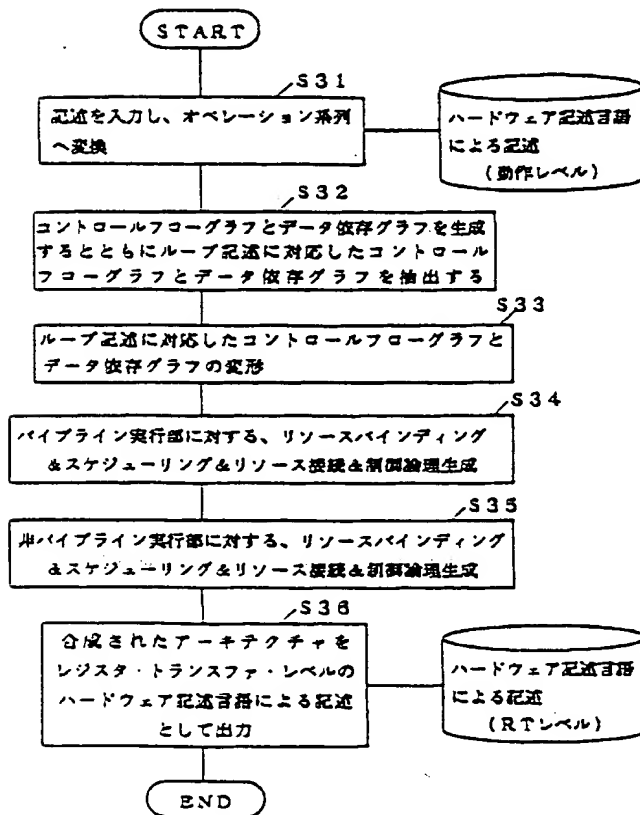
【図15】



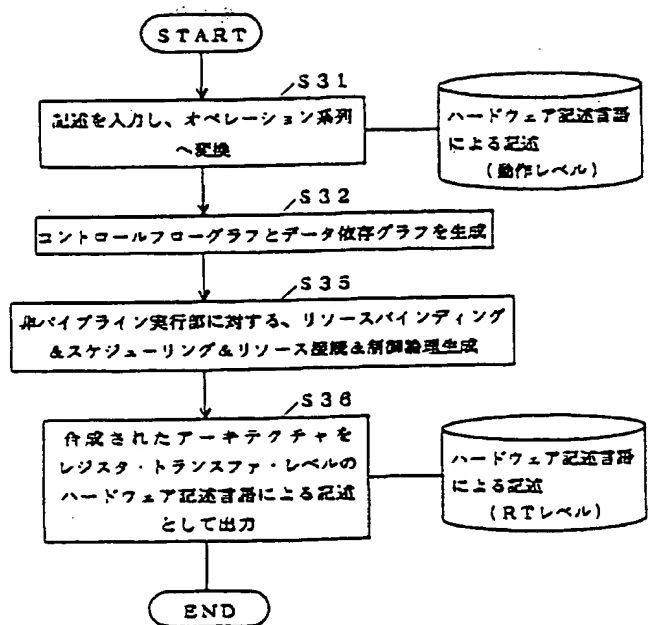
【図12】



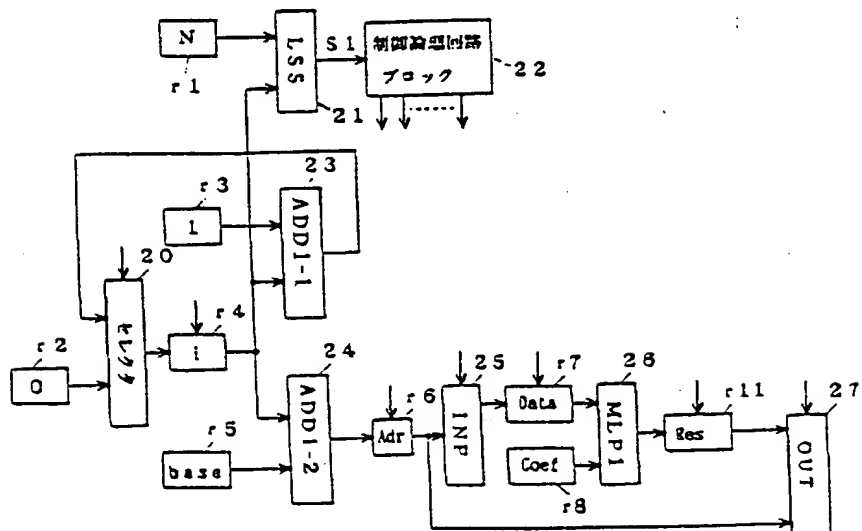
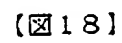
【図13】



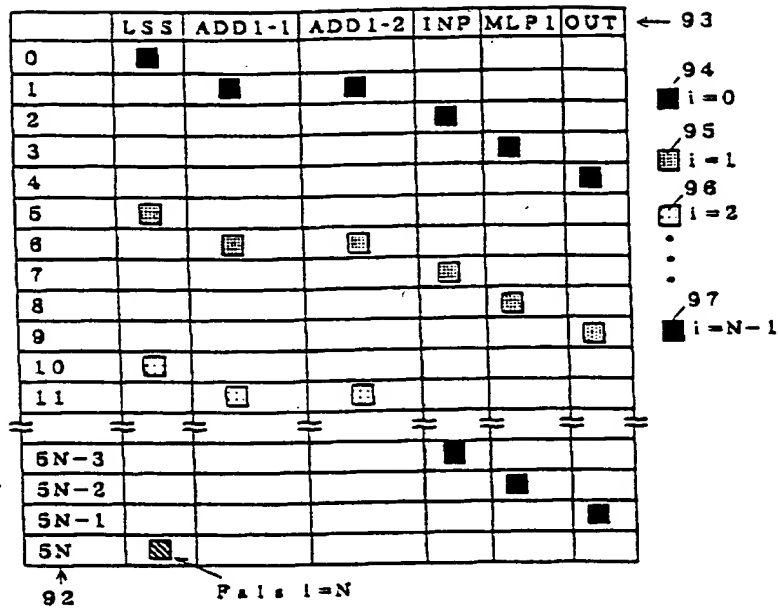
【図16】



START



【図19】



【図20】

